JBoss Communications Platform 5.1

JAIN SLEE User Guide

The Guide to the JAIN SLEE Server

Alexandre Mendonça

Amit Bhayani

Bartosz Baranowski

Eduardo Martins

Ivelin Ivanov

JBoss Communications Platform 5.1 JAIN SLEE User Guide
The Guide to the JAIN SLEE Server
Edition 5.1.0

Author	Alexandre Mendonça	*amendonca@redhat.com*
Author	Amit Bhayani	*abhayani@redhat.com*
Author	Bartosz Baranowski	*bbaranow@redhat.com*
Author	Eduardo Martins	*emartins@redhat.com*
Author	Ivelin Ivanov	*iivanov@redhat.com*

The JBoss Communications Platform (JBCP) is the first and *only* open source VoIP platform certified for JAIN SLEE 1.1 and SIP Servlets 1.1 compliance. JBCP serves as a high performance core for Service Delivery Platforms (SDPs) and IP Multimedia Subsystems (IMSs) by leveraging J2EE to enable the convergence of data and video in Next-Generation Intelligent Network (NGIN) applications.

Preface

1. Document Conventions

This manual uses several conventions to highlight certain words and phrases and draw attention to specific pieces of information.

In PDF and paper editions, this manual uses typefaces drawn from the *Liberation Fonts*[1] set. The Liberation Fonts set is also used in HTML editions if the set is installed on your system. If not, alternative but equivalent typefaces are displayed. Note: Red Hat Enterprise Linux 5 and later includes the Liberation Fonts set by default.

1.1. Typographic Conventions

Four typographic conventions are used to call attention to specific words and phrases. These conventions, and the circumstances they apply to, are as follows.

`Mono-spaced Bold`

Used to highlight system input, including shell commands, file names and paths. Also used to highlight keycaps and key combinations. For example:

> To see the contents of the file **`my_next_bestselling_novel`** in your current working directory, enter the **`cat my_next_bestselling_novel`** command at the shell prompt and press **`Enter`** to execute the command.

The above includes a file name, a shell command and a keycap, all presented in mono-spaced bold and all distinguishable thanks to context.

Key combinations can be distinguished from keycaps by the hyphen connecting each part of a key combination. For example:

> Press **`Enter`** to execute the command.
>
> Press **`Ctrl`**+**`Alt`**+**`F2`** to switch to the first virtual terminal. Press **`Ctrl`**+**`Alt`**+**`F1`** to return to your X-Windows session.

The first paragraph highlights the particular keycap to press. The second highlights two key combinations (each a set of three keycaps with each set pressed simultaneously).

If source code is discussed, class names, methods, functions, variable names and returned values mentioned within a paragraph will be presented as above, in **`mono-spaced bold`**. For example:

> File-related classes include **`filesystem`** for file systems, **`file`** for files, and **`dir`** for directories. Each class has its own associated set of permissions.

Proportional Bold

This denotes words or phrases encountered on a system, including application names; dialog box text; labeled buttons; check-box and radio button labels; menu titles and sub-menu titles. For example:

> Choose **System** → **Preferences** → **Mouse** from the main menu bar to launch **Mouse Preferences**. In the **Buttons** tab, click the **Left-handed mouse** check box and click

[1] https://fedorahosted.org/liberation-fonts/

> **Close** to switch the primary mouse button from the left to the right (making the mouse suitable for use in the left hand).
>
> To insert a special character into a **gedit** file, choose **Applications** → **Accessories** → **Character Map** from the main menu bar. Next, choose **Search** → **Find…** from the **Character Map** menu bar, type the name of the character in the **Search** field and click **Next**. The character you sought will be highlighted in the **Character Table**. Double-click this highlighted character to place it in the **Text to copy** field and then click the **Copy** button. Now switch back to your document and choose **Edit** → **Paste** from the **gedit** menu bar.

The above text includes application names; system-wide menu names and items; application-specific menu names; and buttons and text found within a GUI interface, all presented in proportional bold and all distinguishable by context.

`Mono-spaced Bold Italic` or ***`Proportional Bold Italic`***

Whether mono-spaced bold or proportional bold, the addition of italics indicates replaceable or variable text. Italics denotes text you do not input literally or displayed text that changes depending on circumstance. For example:

> To connect to a remote machine using ssh, type **`ssh`** ***`username@domain.name`*** at a shell prompt. If the remote machine is **`example.com`** and your username on that machine is john, type **`ssh john@example.com`**.
>
> The **`mount -o remount`** ***`file-system`*** command remounts the named file system. For example, to remount the **`/home`** file system, the command is **`mount -o remount /home`**.
>
> To see the version of a currently installed package, use the **`rpm -q`** ***`package`*** command. It will return a result as follows: ***`package-version-release`***.

Note the words in bold italics above — username, domain.name, file-system, package, version and release. Each word is a placeholder, either for text you enter when issuing a command or for text displayed by the system.

Aside from standard usage for presenting the title of a work, italics denotes the first use of a new and important term. For example:

> Publican is a *DocBook* publishing system.

1.2. Pull-quote Conventions

Terminal output and source code listings are set off visually from the surrounding text.

Output sent to a terminal is set in **`mono-spaced roman`** and presented thus:

```
books        Desktop   documentation  drafts  mss    photos   stuff  svn
books_tests  Desktop1  downloads      images  notes  scripts  svgs
```

Source-code listings are also set in **`mono-spaced roman`** but add syntax highlighting as follows:

```
package org.jboss.book.jca.ex1;

import javax.naming.InitialContext;
```

```
public class ExClient
{
   public static void main(String args[])
       throws Exception
   {
      InitialContext iniCtx = new InitialContext();
      Object         ref    = iniCtx.lookup("EchoBean");
      EchoHome       home   = (EchoHome) ref;
      Echo           echo   = home.create();

      System.out.println("Created Echo");

      System.out.println("Echo.echo('Hello') = " + echo.echo("Hello"));
   }
}
```

1.3. Notes and Warnings

Finally, we use three visual styles to draw attention to information that might otherwise be overlooked.

Note

Notes are tips, shortcuts or alternative approaches to the task at hand. Ignoring a note should have no negative consequences, but you might miss out on a trick that makes your life easier.

Important

Important boxes detail things that are easily missed: configuration changes that only apply to the current session, or services that need restarting before an update will apply. Ignoring a box labeled 'Important' will not cause data loss but may cause irritation and frustration.

Warning

Warnings should not be ignored. Ignoring warnings will most likely cause data loss.

2. We Need Feedback

If you find a typographical error in this manual, or if you have thought of a way to make this manual better, submit a report in Bugzilla: *http://bugzilla.redhat.com/bugzilla/*, against the **JBoss Communication Platform**.

When submitting a bug report, be sure to mention the manual's identifier: *doc-JAIN_SLEE_User_Guide*.

If you have a suggestion for improving the documentation, try to be as specific as possible when describing it. If you have found an error, please include the section number and some of the surrounding text so we can find it easily.

Introduction to JBCP JAIN SLEE

JAIN SLEE is the specification for a Java Service Logic and Execution Environment (SLEE) architecture, created in the Java Community Process (JCP) by several individuals and companies, including Red Hat. A SLEE is an application server, or service container, which defines a component model for structuring the logic of communications services as a collection of reusable components, and for combining these components into even more sophisticated services. This model was designed and optimized for event-driven applications.

In addition to the service component model, the SLEE also defines management interfaces used to administer the container and the service components executing within, and a set of standard facilities, which provide common features, such as timers, traces and alarms, to JAIN SLEE components.

JBCP JAIN SLEE is the first and only open source platform certified for JAIN SLEE 1.1 compliance, providing a highly scalable, event-driven application server with a robust component model and a fault tolerant execution environment.

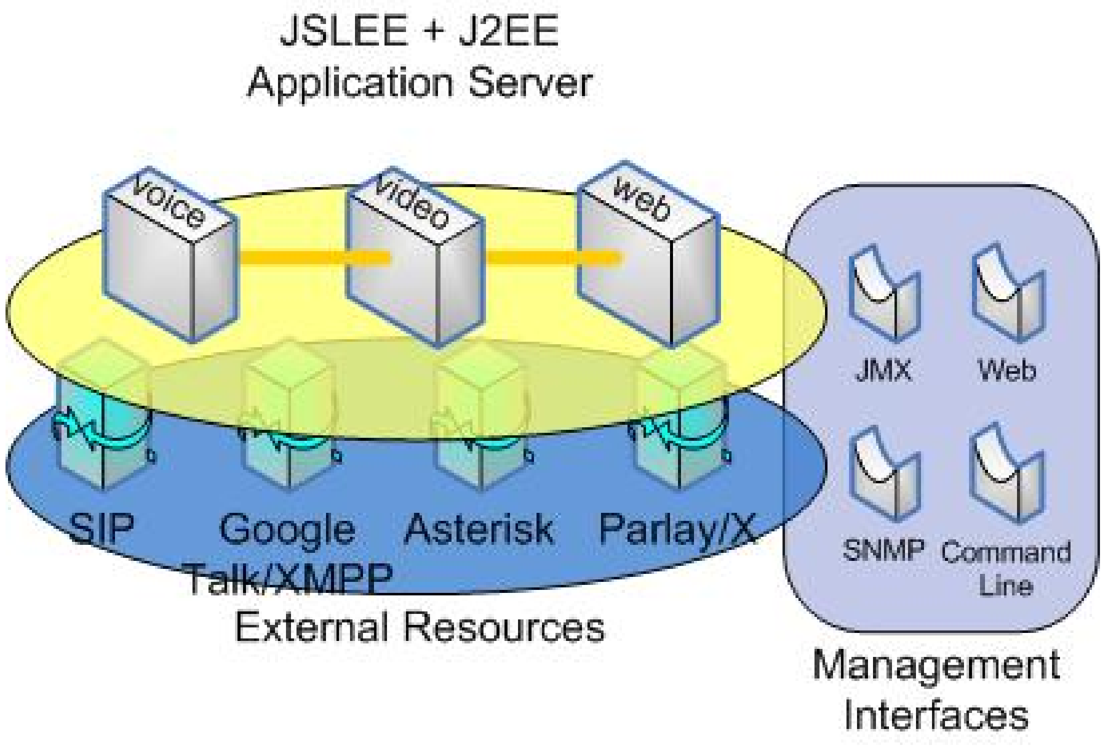

Figure 1.1. Overview of JAIN SLEE

The JBoss Communications Platform JAIN SLEE is built on top of the open source award winning JBoss Enterprise Application Platform, which means that JBoss Communications Platform JAIN SLEE complements JAIN SLEE with Java Enterprise (JEE) 5 container features, allowing strong convergence of different application models, for even feature richer communication services, for instance, the Web and SIP can be combined to achieve a more sophisticated and natural user experience. JBoss Communications Platform JAIN SLEE inherits quality management features and tools from JBoss Enterprise Application Platform, such as the JMX Console, JBoss Operations Network Plugins and SNMP Adaptor.

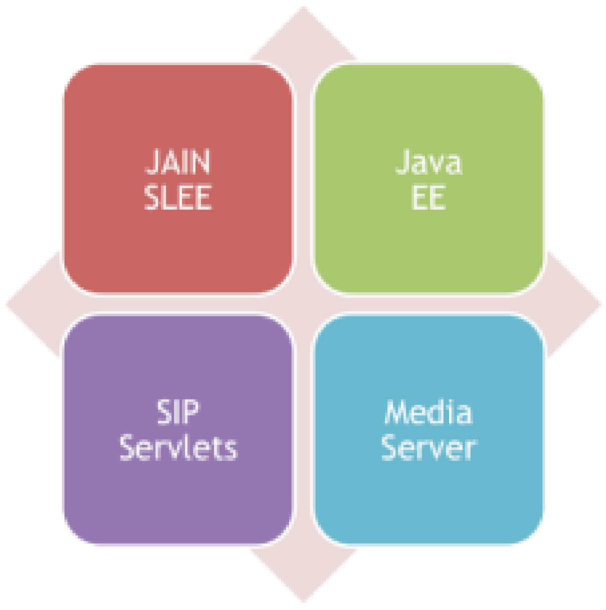

Figure 1.2. JBoss Communications Platform

JBoss Communications Platform JAIN SLEE can also be complemented with JBCP SIP Servlets and JBoss Communications Platform Media Server, providing unique value and integration features not found elsewhere.

Installing JAIN SLEE

The JAIN SLEE Server is installed as part of the JBoss Communications Platform. For system requirements, configuration options, and installation instructions, please refer to the *Platform Installation Guide*[1].

[1] http://docs.redhat.com/docs/en-US/JBoss_Communications_Platform/5.1/html-single/Platform_Installation_Guide/index.html

Configuring and Running JAIN SLEE

3.1. Server Profiles

JBoss Communications Platform JAIN SLEE reuses JBoss Enterprise Application Platform server profiles to expose different configurations for different needs:

Default Profile
: The **default** profile is proper for standalone or pure high availability. It provides the best performance per cluster node, with linear scaling, but there is no state replication in the cluster, which means that there is no support for failover, neither there is any kind of state gravitation (one node sending state so another node continues its work.

All Profile
: The **all** profile is proper for more flexible high availability and failover support. Performance per node decreases but the cluster does state replication or gravitation.

Profiles can be selected when starting the server, see *Section 3.2, “Running JAIN SLEE”* for detailed instructions.

3.2. Running JAIN SLEE

Starting or stopping JBoss Communications Platform JAIN SLEE is no different than starting or stopping JBoss Enterprise Application Platform

3.2.1. Starting

Once installed, you can run server(s) by executing the run.sh startup scripts in the ***`$JBCP_ROOT`*** **`/bin`** directory.

3.2.1.1. Starting Parameters

Server Profile
: To specify the server profile use **`-c profile_name`**, for instance, to use the **all** profile then start the server with **`-c all`**

> **Important**
>
> If not specified the **default** profile is used.

IP / Host
: To specify the IP/Host which the server binds, use **`-b IP`**, for instance, to use the 192.168.0.1 IP then start the server with **`-b 192.168.0.1`**

> **Important**
>
> If not specified then **127.0.0.1** is used.

3.2.2. Stopping

You can shut down the server(s) you can run server(s) by executing the **`shutdown.sh -s`** script in the ***`$JBCP_ROOT`*`/bin`** directory. Note that if you properly stop the server, you will see the following three lines as the last output in the Unix terminal or Command Prompt:

```
[Server] Shutdown complete
Shutdown complete
Halting VM
```

3.3. Configuring JAIN SLEE

JAIN SLEE is configured through an XML descriptor for each *Section 3.1, “Server Profiles”*. The XML file is named **`jboss-beans.xml`** and is located at ***`$JBCP_ROOT`*`/server/`*`$PROFILE`*`/deploy/mobicents-slee/META-INF`**, where **profile_name** is the server profile name.

Warning

This configuration greatly affects performance or correctness of the container behavior. This is for advanced users that know the internals of the container.

3.3.1. Event Router Statistics and Configuration

The JAIN SLEE Event Router is the module responsible for creating new service instances and delivering events to all interested parties. It is capable of doing the routing of several events in parallel, through the usage of multiple executors, each bound to a different thread.

The Event Router is also able to account performance and load statistics, indicating the number of activities being assigned or several timings regarding event routing, globally or for each individual executor/thread. Statistics are turned on by default and may be retrieved through the JMX MBean **org.mobicents.slee:name=EventRouterStatistics**.

An important sub-module of the Event Router is the Executor Mapper, which is responsible for assigning activities to the available executors. JAIN SLEE includes two different Executor Mappers. The default one takes into account the hashcode of the activity handle when distributing, while the alternative uses a round robin algorithm.

Note

In the case of advanced performance tuning, it is advised to try the different implementations available, or even provide a custom one.

The Executor Mapper is nothing more than an interface: **`org.mobicents.slee.container.eventrouter.EventRouterExecutorMapper`**. To deploy a custom implementation, drop the implementation class or classes, packed in a jar file, in the server profile **`/deploy`** directory.

The whole Event Router is a critical component with respect to the container's performance. Its configuration can be tuned, through an XML file and a JMX MBean.

3.3.1.1. Event Router Persistent Configuration

Configuration is done through an XML descriptor for each *Section 3.1, “Server Profiles”*. The XML file is named **`jboss-beans.xml`** and is located at ***`$JBCP_ROOT`*`/server/`*`$PROFILE`*`/deploy/mobicents-slee/META-INF`**.

The configuration is exposed a JBoss Microcontainer Bean:

```
<bean name="Mobicents.JAINSLEE.EventRouterConfiguration"
 class="org.mobicents.slee.container.management.jmx.EventRouterConfiguration">
 <annotation>@org.jboss.aop.microcontainer.aspects.jmx.JMX(name=
  "org.mobicents.slee:name=EventRouterConfiguration", exposedInterface=
  org.mobicents.slee.container.management.jmx.EventRouterConfigurationMBean.class,
  registerDirectly=true)</annotation>
 <property name="eventRouterThreads">8</property>
 <property name="collectStats">true</property>
 <property name="executorMapperClassName">
  org.mobicents.slee.runtime.eventrouter.mapping.ActivityHashingEventRouterExecutorMapper
 </property>
</bean>
```

Table 3.1. JAIN SLEE Event Router Bean Configuration

Property Name	Property Type	Description
eventRouterThreads	int	defines how many executors should be used by the Event Router, each bounds to a different thread
collectStats	boolean	defines if performance and load statistics should be collected, turning this feature off will increase performance
confirmSbbEntityAttachement	boolean	defines if the event router should reconfirm that sbb entities are attached to activity context, before delivering event, this will avoid that a sbb entity handles concurrent events after it detachs, turning this feature off will increase performance
executorMapperClassName	Class	This property defines the implementation class of Executor Mapper used by the Event Router, the one above and default uses the activity handle hashcode to do the mapping, an alternative is **`org.mobicents.slee.runtime.eventro`** which uses Round Robin algorithm.

3.3.1.2. Event Router JMX Configuration

Through JMX, the Event Router module configuration can be changed while the container is running. These configuration changes are not persisted.

The JMX MBean that can be used to change the Event Router configuration is named **org.mobicents.slee:name=EventRouterConfiguration**, and provides getters and setters to change each property defined in the persistent configuration. See *Section 4.3.1, “JMX Console”* for how the JMX Console can be used to use this MBean.

3.3.2. Timer Facility Configuration

The JAIN SLEE Timer Facility is the module responsible for managing SLEE timers. The number of threads the facility uses is configurable.

The Timer Facility configuration can be changed through an XML file and a JMX MBean.

3.3.2.1. Timer Facility Persistent Configuration

Each JBoss Communications Platform cluster is configured through an xml file called **`jboss-beans.xml`**, located in the ***`$JBCP_ROOT`*****`/server/`*****`$PROFILE`*****`/deploy/mobicents-slee/META-INF`** directory.

A JBoss Microcontainer Bean exposes the configuration options:

```
<bean name="Mobicents.JAINSLEE.TimerFacilityConfiguration"
 class="org.mobicents.slee.container.management.jmx.TimerFacilityConfiguration">
 <annotation>@org.jboss.aop.microcontainer.aspects.jmx.JMX(name=
  "org.mobicents.slee:name=TimerFacilityConfiguration",exposedInterface=
  org.mobicents.slee.container.management.jmx.TimerFacilityConfigurationMBean.class,
  registerDirectly=true)</annotation>
 <property name="timerThreads">4</property>
</bean>
```

There is only one property that can be configured: *`timerThreads`*. This property is represented by an integer, and defines how many threads should be used by the Timer Facility.

3.3.2.2. Timer Facility JMX Configuration

The Timer Facility module configuration can be changed while the container is running, through the JMX console.

JMX Configuration Changes

Configuration changes made in the JMX console are not persisted.

The Timer Facility can be configured through the JMX console, using the **`org.mobicents.slee:name=TimerFacilityConfiguration`** MBean. See *Section 4.3.1, “JMX Console”* for how to use the JMX console.

3.3.3. Configuring JAIN SLEE Profiles

JAIN SLEE Profiles is a component used to store data, usually related with a user and/or service profile. JAIN SLEE maps JAIN SLEE Profiles to a Java Persistence API (JPA) Datasource, through Hibernate.

There are two configurations for JAIN SLEE Profiles provided as JBoss Microcontainer Beans:

```
<bean name="Mobicents.JAINSLEE.Profiles.JPA.HSQLDBConfig"
 class="org.mobicents.slee.container.deployment.profile.jpa.Configuration">
 <property name="persistProfiles">true</property>
 <property name="clusteredProfiles">false</property>
 <property name="hibernateDatasource">java:/DefaultDS</property>
 <property name="hibernateDialect">org.hibernate.dialect.HSQLDialect</property>
 <depends>jboss.jca:service=DataSourceBinding,name=DefaultDS</depends>
</bean>
<bean name="Mobicents.JAINSLEE.Profiles.JPA.PostgreSQLConfig"
 class="org.mobicents.slee.container.deployment.profile.jpa.Configuration">
 <property name="persistProfiles">true</property>
 <property name="clusteredProfiles">true</property>
 <property name="hibernateDatasource">java:/PostgresDS</property>
 <property name="hibernateDialect">org.hibernate.dialect.PostgreSQLDialect</property>
</bean>
```

Important

Configurations can be changed, or new ones can be added. For new ones, ensure that the name attribute of the bean element is unique.

Table 3.2. JAIN SLEE Profiles Bean Configuration

Property Name	Property Type	Description
persistProfiles	boolean	If true, profile changes are persisted into the data source.
clusteredProfiles	boolean	If true, then the container is aware there is a shared data source and that updates may be done by other nodes (for example, deletion of a JAIN SLEE profile table).
hibernateDatasource	String	The name of the Java Datasource deployed in the JBoss Enterprise Application Platform.
hibernateDialect	String	The java class name of the hibernate dialect to use, related with the selected datasource.

To switch the active configuration simply change the parameter injected in the bean named **Mobicents.JAINSLEE.Container**.

3.3.4. Configuring JAIN SLEE Usage Params

JAIN SLEE Usage Params are usage statistics of several JAIN SLEE concepts, such as the SBB, Service, Resource Adaptor or Profile Table. Configuration for JAIN SLEE Usage Params is done through a JBoss Microcontainer Bean:

```
<bean name="Mobicents.JAINSLEE.UsageMbeans.Config"
 class="org.mobicents.slee.runtime.usage.UsageMBeansConfiguration">
 <property name="clusteredUsageMBeans">false</property>
</bean>
```

There is a single property to configure, named clusteredUsageMBeans, with a boolean value. Set this property to *`true`* if the usage param values should be shared by all cluster nodes. This option means a significant performance penalty, but makes it possible to learn the current values for the whole cluster by checking a single node. Use this only if there is no other solution, such as a management client, that gets the value of each node and then does the appropriate calculations.

3.3.5. Other Configurations

Other JAIN SLEE runtime configuration is done through the following JBoss Microcontainer Bean:

```
<bean name="Mobicents.JAINSLEE.MobicentsManagement"
 class="org.mobicents.slee.container.management.jmx.MobicentsManagement">
 <annotation>@org.jboss.aop.microcontainer.aspects.jmx.JMX(
  name="org.mobicents.slee:service=MobicentsManagement",
  exposedInterface=org.mobicents.slee.container.management.
  jmx.MobicentsManagementMBean.class,
  registerDirectly=true)</annotation>
 <property name="entitiesRemovalDelay">1</property>
 <property name="timerThreads">8</property>
 <property name="loadClassesFirstFromAS">true</property>
</bean>
```

Table 3.3. Other JAIN SLEE Configurations

Property Name	Property Type	Description
entitiesRemovalDelay	int	The number of minutes before the container forces the ending of SBB entities from a service being deactivated.
timerThreads	int	The number of threads used by the timer facility.
loadClassesFirstFromAS	boolean	If true, a classloader always looks for classes first in the container's own classloader, which shares and imports classes from JBoss Enterprise Application Platform.

This configuration can be changed with the container running with JMX. Note that such configuration changes are not persisted.

To change the configuraton, use the JMX MBean named **org.mobicents.slee:service=MobicentsManagement**, which provides getters and setters to change each property defined in the persistent configuration that is configurable with the container running. The JMX Console can be used to use this MBean, as described in *Section 4.3.1, “JMX Console”*.

3.3.6. Logging Configuration

Logging configuration is documented in section *Section 5.1.1, “Simplified Global Log4j Configuration”*

3.3.7. Congestion Control Configuration

Congesture Control feature configuration is documented in section *Section 9.2.1, “Congestion Control Configuration”*

Managing JAIN SLEE

4.1. JAIN SLEE JMX Agent

The JMX Agent exposes all MBeans running in the server, including those mandated by the JAIN SLEE 1.1 specification. By default, the Agent listens to port 1099. It is only available at the host/ip the server is bound to.

Important

Operations completed through the JMX Agent are not carried over once the server is shutdown. For example, if a deployable unit is installed through JMX, and the server is then shutdown, the deployable unit will no longer be installed once the server is started again.

4.2. Managing JAIN SLEE Components

4.2.1. Persistent Deployable Unit Management

JAIN SLEE provides a file deployer that greatly simplifies the management of JAIN SLEE deployable unit jars. The deployer:

- Handles the installation of enclosed JAIN SLEE components.
- Automatically activates and deactivates services contained.
- Handles the creation, removal, activation, deactivation, link binding and link unbinding of all Resource Adapter Entities.

All operations are persistent, which means that unlike management done through JMX, these survive server shutdowns.

4.2.1.1. Persistent Deployable Unit Install

To install a deployable unit jar simply copy the file to ***$JBCP_ROOT*/server/*$PROFILE*/deploy/**. Child directories can be used.

4.2.1.2. Persistent Deployable Unit Uninstall

To uninstall a deployable unit jar simply delete the file.

4.2.1.3. Beyond Deployable Unit (Un)Install

The file deployer provides additional behavior then simply (un)install deployable unit jars, as done through the JAIN SLEE 1.1 DeploymentMBean:

Service (De)Activation
: All services contained in the deployable unit jar are activated after the install, and deactivated prior to uninstall, or server shutdown.

Dependencies Management
: The deployer puts the installation process on hold until all of the component's dependencies are installed and activated. When uninstalling, it waits for all of the components which depend on

components inside the deployable unit to be uninstalled. After an install or uninstall, the deployer evaluates all operations on hold.

4.2.1.4. Deploy-Config Extension

A deployable unit jar may include a deploy-config.xml file in its **`META-INF/`** directory. This file provides additional management actions for persistent install/uninstall operations:

Resource Adaptor Entity Management
: It is possible to specify RA entities, and the container will create and activate the RA entities after the deployable unit is installed. During the uninstall process, the container will deactivate and remove those RA entities.

Resource Adaptor Links Management
: It is possible to specify RA links, and the container will bind those after the deployable unit is installed. When uninstalled, the container will unbind those RA links as well. The links are set for resource adapter entities created in the **`deploy-config.xml`** file.

This file should comply with the following schema:

```
<?xml version="1.0" encoding="UTF-8" ?>

<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="deploy-config">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="ra-entity" maxOccurs="unbounded" minOccurs="1"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>

  <xs:element name="property">
    <xs:complexType>
      <xs:attribute name="name" type="xs:string" use="required" />
      <xs:attribute name="type" type="xs:string" use="required" />
      <xs:attribute name="value" type="xs:string" use="required" />
    </xs:complexType>
  </xs:element>

  <xs:element name="properties">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="property" maxOccurs="unbounded" minOccurs="0"/>
      </xs:sequence>
      <xs:attribute name="file" type="xs:string" use="optional" />
    </xs:complexType>
  </xs:element>

  <xs:element name="ra-entity">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="properties" maxOccurs="1" minOccurs="0"/>
        <xs:element ref="ra-link" maxOccurs="unbounded" minOccurs="0"/>
      </xs:sequence>
      <xs:attribute name="resource-adaptor-id" type="xs:string" use="required" />
      <xs:attribute name="entity-name" type="xs:string" use="required" />
    </xs:complexType>
  </xs:element>

  <xs:element name="ra-link">
    <xs:complexType>
      <xs:attribute name="name" type="xs:string" use="required" />
    </xs:complexType>
  </xs:element>

</xs:schema>
```

```

<ra-entity
 resource-adaptor-
id="ResourceAdaptorID[name=JainSipResourceAdaptor,vendor=net.java.slee.sip,version=1.2]"
 entity-name="SipRA">
 <properties>
  <property name="javax.sip.PORT" type="java.lang.Integer" value="5060" />
 </properties>
 <ra-link name="SipRA" />
</ra-entity>
```

The deploy-config.xml example above defines a resource adaptor entity named **`SipRa`**, to be created for the resource adaptor with id **`ResourceAdaptorID[name=JainSipResourceAdaptor, vendor=net.java.slee.sip, version=1.2]`**, and with a single config property named **`javax.sip.PORT`** of type **`java.lang.Integer`** and with value **`5060`**. Additionally, a resource adaptor link named **`SipRa`** should be bound to the resource adaptor entity.

After the deployable unit is installed, the resource adaptor entity is created, activated and the resource adaptor link is bound. Before the deployable unit is uninstalled, or the server is shutdown, the link is unbound, then the resource adaptor entity is deactivated, and finally the same resource adaptor entity is removed.

4.2.2. Ant Tasks

JAIN SLEE includes some tasks for Apache Ant, which can be used for common management tasks done through JMX, when the container is running. The tasks come bundled in ***`$JBCP_ROOT/`*****`server/`*****_`$PROFILE`_****`/deploy/mobicents-slee/lib/ant-tasks.jar`**. To use these, the Ant script must include the following code:

```

<property environment="system" />
<property name="node" value="default" />
<property name="jboss.deploy" value="${system.JBOSS_HOME}/server/default/deploy" />

<path id="project.classpath">
 <fileset dir="${jboss.deploy}/mobicents-slee/lib">
  <include name="*.jar" />
 </fileset>
 <fileset dir="${system.JBOSS_HOME}/client">
  <include name="*.jar" />
 </fileset>
</path>

<property name="project.classpath" refid="project.classpath" />

<property name="jnpHost" value="127.0.0.1" />
<property name="jnpPort" value="1099" />

<taskdef name="slee-management"
 classname="org.mobicents.ant.MobicentsManagementAntTask"
 classpath="${project.classpath}" />
```

It is important to understand some properties set by the code above:

node
: This property defines the server configuration profile to be used, for further information about those refer to *Section 3.1, “Server Profiles”*.

jnpHost
: The host/ip which JBoss Communications Platform JAIN SLEE is bound.

jnpPort
: The port which the JMX Agent is listening.

Important

The property values can be overridden when invoking the Ant script. To do this, use the parameter *-DpropertyName=propertyValue*. For example, the server profile can be changed from **default** to **all** using *-Dnode=all*.

4.2.2.1. SLEE Management Task

The **slee-management** task allows a script to manage JAIN SLEE deployable units, services and resource adapters. The operations, or sub-tasks, are done through JMX.

```
<slee-management jnpport="${jnpPort}" host="${jnpHost}">
 <!-- sub-tasks -->
</slee-management>
```

The attributes have the same meaning as the properties listed in the script code to import the tasks here: *Section 4.2.2, “Ant Tasks”*.

4.2.2.1.1. Install Deployable Unit Sub-Task

The **install** sub-task installs JAIN SLEE the deployable unit jar pointed by the value of attribute *url*. Example of usage:

```
<slee-management jnpport="${jnpPort}" host="${jnpHost}">
 <install url="file:///tmp/my-deployable-unit.jar" />
</slee-management>
```

4.2.2.1.2. Uninstall Deployable Unit Sub-Task

The **uninstall** sub-task uninstalls JAIN SLEE the deployable unit jar which was installed from the value of attribute *url*. Example of usage:

```
<slee-management jnpport="${jnpPort}" host="${jnpHost}">
 <uninstall url="file:///tmp/my-deployable-unit.jar" />
</slee-management>
```

4.2.2.1.3. Activate Service Sub-Task

The **activateservice** sub-task activates an already installed JAIN SLEE service, with the id specified by the value of attribute *componentid*. Example of usage:

```
<slee-management host="${jnpHost}" jnpport="${jnpPort}">
 <activateservice
  componentid="ServiceID[name=FooService,vendor=org.mobicents,version=1.0]" />
</slee-management>
```

4.2.2.1.4. Deactivate Service Sub-Task

The **`deactivateservice`** sub-task deactivates an already installed JAIN SLEE service, with the id specified by the value of attribute *`componentid`*. Example of usage:

```
<slee-management host="${jnpHost}" jnpport="${jnpPort}">
 <deactivateservice
  componentid="ServiceID[name=FooService,vendor=org.mobicents,version=1.0]" />
</slee-management>
```

4.2.2.1.5. Create Resource Adaptor Entity Sub-Task

The **`createraentity`** sub-task creates the resource adaptor entity with the name specified by the value of attribute *`entityname`*, for an already installed JAIN SLEE resource adaptor, with the id specified by the value of attribute *`componentid`*. Example of usage:

```
<slee-management host="${jnpHost}" jnpport="${jnpPort}">
 <createraentity
  componentid="ResourceAdaptorID[name=FooRA,vendor=org.mobicents,version=1.0]"
  entityname="FooRA" />
</slee-management>
```

4.2.2.1.6. Remove Resource Adaptor Entity Sub-Task

The **`removeraentity`** sub-task removes the resource adaptor entity with the name specified by the value of attribute *`entityname`*. Example of usage:

```
<slee-management host="${jnpHost}" jnpport="${jnpPort}">
 <removeraentity entityname="FooRA" />
</slee-management>
```

4.2.2.1.7. Activate Resource Adaptor Entity Sub-Task

The **`activateraentity`** sub-task activates the resource adaptor entity with the name specified by the value of attribute *`entityname`*. Example of usage:

```
<slee-management host="${jnpHost}" jnpport="${jnpPort}">
 <activateraentity entityname="FooRA" />
</slee-management>
```

4.2.2.1.8. Deactivate Resource Adaptor Entity Sub-Task

The **`deactivateraentity`** sub-task deactivates the resource adaptor entity with the name specified by the value of attribute *`entityname`*. Example of usage:

```
<slee-management host="${jnpHost}" jnpport="${jnpPort}">
 <deactivateraentity entityname="FooRA" />
</slee-management>
```

4.2.2.1.9. Bind Resource Adaptor Link Sub-Task

The **`bindralinkname`** sub-task binds the resource adaptor link with the name specified by the value of attribute *`linkname`*, for an already active JAIN SLEE resource adaptor entity, with the name specified by the value of attribute *`entityname`*. Example of usage:

```
<slee-management host="${jnpHost}" jnpport="${jnpPort}">
 <bindralinkname entityname="FooRA"
  linkname="FooRA" />
</slee-management>
```

4.2.2.1.10. Unbind Resource Adaptor Link Sub-Task

The **`unbindralinkname`** sub-task unbinds the resource adaptor link with the name specified by the value of attribute *`linkname`*. Example of usage:

```
<slee-management host="${jnpHost}" jnpport="${jnpPort}">
 <unbindralinkname linkname="FooRA" />
</slee-management>
```

4.3. Management Consoles

4.3.1. JMX Console

JBoss Enterprise Application Platform provides a simple web console that gives quick access to all MBeans registered to the server, including those defined by the JAIN SLEE 1.1. specification.

Procedure 4.1. Accessing the JMX console

1. Start the server. Refer to the *Platform Installation Guide*[1] for instructions for running the server.

2. Point a web browser to **`http://ip:8080/jmx-console`**, where **`ip`** is the IP/Host that the container is bound to. Unless set during start up, the IP/Host will default to **`127.0.0.1/localhost`**.

MBeans in the **`javax.slee`** domain are all standard JAIN SLEE 1.1 MBeans, while the ones in the **`org.mobicents.slee`** domain are proprietary to JBoss Communications Platform JAIN SLEE. The following MBeans are of particular interest:

`org.mobicents.slee:service=MobicentsManagement`
: This MBean can be used to make non-persistent changes to the server configuration, in runtime. The **`dumpContainerState`** operation displays a textual snapshot of the server's state, which can be used to quickly look for memory leaks or other debug/profiling related tasks.

`org.mobicents.slee:name=DeployerMBean`
: the MBean allows interaction with the persistent deployable unit deployer. The operation **`showStatus`** displays a textual snapshot of the deployers's state, which can be used to quickly find out if there is any deployable unit deployment pending, for instance, due to missing dependencies.

`org.mobicents.slee:name=CongestionControlConfiguration`
: the MBean allows changing or retrieving the Congestion Control feature, with the container running. Details are provided in section *Section 9.2.1, “Congestion Control Configuration”*.

Important

For further information about JAIN SLEE 1.1 MBeans and their operations refer to the JAIN SLEE 1.1 Specification, all are covered with great detail.

4.3.2. JBoss Operations Network Console

The JMX Console is simple but the MBeans operations were made considering its invocation by management clients, not people using browsers. JBoss Operations Network was developed to become Red Hat's Middleware Administration Tool, providing an unified interface and extensible model, to be used to control, monitor and manage servers individually, and clusters.

JBoss Communications Platform JAIN SLEE binary release includes a JBoss Operations Network Console in **`tools/management-console`**, with standalone documentation.

4.3.3. `TWIDDLE` CLI

Both, JMX Console and JBoss Operations Network Console, are graphic(web) based tools. Some deployments may require command line access to JBCP SLEE. To aid such cases, JBCP offers **`TWIDDLE`** based CLI. It allows to manage single instance (remote or local) of the JBCP SLEE server.

JBoss Communications Platform JAIN SLEE binary release includes a **`TWIDDLE`** CLI in **`tools/twiddle`**, with standalone documentation.

Logging, Traces and Alarms

5.1. Log4j Logging Service

In JBCP JAIN SLEE **Apache log4j** is used for logging. If you are not familiar with the log4j package and would like to use it in your applications, you can read more about it at the *Jakarta web site*[1].

Logging is controlled from a central **conf/jboss-log4j.xml** file, in each server configuration profile. This file defines a set of appenders specifying the log files, what categories of messages should go there, the message format and the level of filtering. By default, the JBoss Communications Platform produces output to both the console and a log file (**log/server.log**).

There are 6 basic log levels used: TRACE, DEBUG, INFO, WARN, ERROR and FATAL.

Logging is organized in categories and appenders. Appenders control destination of log entries. Different appenders differ in configuration, however each supports threshold. Threshold filters log entries based on their level. Threshold set to WARN will allow log entry to pass into appender if its level is WARN, ERROR or FATAL, other entries will be discarded. For more details on appender configuration please refer to its documentation or java doc.

The logging threshold on the console is INFO, by default. In contrast, there is no threshold set for the server.log file, so all generated logging messages are logged there.

Categories control level for loggers and its children, for details please refer to log4j manual.

By default JBCP JAIN SLEE inherits level of INFO from root logger. To make platform add more detailed logs, file **conf/jboss-log4j.xml** has to be altered. Explicit category definition for JAIN SLEE looks like:

```
<category name="org.mobicents.slee">
    <priority value="INFO"/>
</category>
```

This limits the level of logging to INFO for all JBoss Communications Platform JAIN SLEE classes. It is possible to declare more categories with different level, to provide logs with greater detail.

For instance, to provide detailed information on JBCP JAIN SLEE transaction engine in separate log file(**txmanager.log**), file **conf/jboss-log4j.xml** should contain entries as follows:

```
<appender name="TXMANAGER" class="org.jboss.logging.appender.RollingFileAppender">
    <errorHandler class="org.jboss.logging.util.OnlyOnceErrorHandler"/>
    <param name="File" value="${jboss.server.home.dir}/log/txmanager.log"/>
    <param name="Append" value="false"/>
    <param name="MaxFileSize" value="500KB"/>
    <param name="MaxBackupIndex" value="1"/>

    <layout class="org.apache.log4j.PatternLayout">
        <param name="ConversionPattern" value="%d %-5p [%c] %m%n"/>
    </layout>
</appender>

<category name="org.mobicents.slee.runtime.transaction">
```

[1] http://jakarta.apache.org/log4j/

```
    <priority value="DEBUG" />
    <appender-ref ref="TXMANAGER"/>
</category>
```

This creates a new file appender and specifies that it should be used by the logger (or category) for the package org.mobicents.slee.runtime.transaction.

The file appender is set up to produce a new log file every day rather than producing a new one every time you restart the server or writing to a single file indefinitely. The current log file is txmanager.log. Older files have the date they were written added to their filenames.

5.1.1. Simplified Global Log4j Configuration

Besides manual logging configuration, described previously, JBoss Communications Platform JAIN SLEE also exposes management operations that greatly simplify such configuration, allowing the administrator to select through predefined and complete logging configuration presets. Such operations are available in MBean named **`org.mobicents.slee%3Aservice%3DMobicentsManagement`**, and the available presets are:

Level

- DEFAULT: Regular logging, at INFO level, displaying most user-related messages;
- DEBUG: More verbose logging, mostly using DEBUG/TRACE level, displaying message of interest for developers;
- PRODUCTION: Low verbosity and async logging, mostly in WARN level, for systems in production so that logging does impact performance.

The available management operations are:

JMX Operation

- getLoggingConfiguration: retrieves what is the current logging configuration;
- switchLoggingConfiguration: allows switching to a different configuration preset;
- setLoggingConfiguration: used to upload a complete logging configuration.

Custom presets can be easily deployed in the application server too. Simply name the configuration file as **`jboss-log4j.xml.PRESET_NAME`**, where **`PRESET_NAME`** should be unique preset name, and copy it to directory ***`$JBCP_ROOT`*`/server/`*`$PROFILE`*`/deploy/mobicents-slee/log4j-templates`**.

Note

These procedures changes the whole JBoss Enterprise Application Platform Platform logging configuration, so it will affect also logging for other running applications besides the JAIN SLEE container.

5.2. Alarm Facility

The **`JAIN SLEE Alarm Facility`** is used by SBBs, Resource Adaptors, and Profiles to request the SLEE to raise or clear alarms. If a request is made to raise an alarm and the identified alarm has not already been raised, the alarm is raised and a corresponding alarm notification is generated by the **`AlarmMBean`**. If a request is made to clear an alarm and the identified alarm is currently raised, the alarm is cleared and a corresponding alarm notification is generated by the **`AlarmMBean`**.

Alarm notifications are intended for consumption by management clients external to the SLEE. The management client is responsible for registering to receive alarm notifications generated by the Alarm Facility through the external management interface of the **`Alarm Facility`**. The management client may optionally provide notification filters so that only the alarm notifications that the management client would like to receive are transmitted to the management client.

For further information on how to use **`JAIN SLEE Alarm Facility`** and receive JMX notifications refer to the JAIN SLEE 1.1 Specification.

5.3. Trace Facility

Notification sources such as SBBs, Resource Adaptors, Profiles, and SLEE internal components can use the **`Trace Facility`** to generate trace messages intended for consumption by external management clients. Management clients register to receive trace messages generated by the **`Trace Facility`** through the external management interface (MBean). Filters can be applied, in a similar way as in case of Alarms.

Within the SLEE, notification sources use a **`tracer`** to emit trace messages. A tracer is a named entity. Tracer names are case-sensitive and follow the Java hierarchical naming conventions. A tracer is considered to be an ancestor of another tracer if its name followed by a dot is a prefix of the descendant tracer's name. A tracer is considered to be a parent of a tracer if there are no ancestors between itself and the descendant tracer. For example, the tracer named **`com`** is the parent tracer of the tracer named **`com.foo`** and an ancestor of the tracer named **`com.foo.bar`**.

All tracers are implicitly associated with a notification source, which identifies the object in the SLEE that is emitting the trace message and is included in trace notifications generated by the **`Trace MBean`** on behalf of the tracer. For instance, an SBB notification source is composed by the SBB id and the Service id.

Important

Multiple notification sources may have tracers with same name in SLEE. Comparing with common logging frameworks, this would mean that the notification source would be part of the log category or name.

For further information on how to use **`JAIN SLEE Trace Facility`** and receive JMX notifications refer to the JAIN SLEE 1.1 Specification.

5.3.1. JAIN SLEE Tracers and Log4j

JAIN SLEE Tracers additionally log messages to **Apache Log4j**, being the log4j category, for notification source **`X`**, defined as **`javax.slee.`** concatenated with the **`X.toString()`**.

For instance, the full log4j logger **name** for tracer named **`GoogleTalkBotSbb`**, of sbb notification source with **`SbbID[name=GoogleTalkBotSbb,vendor=mobicents,version=1.0]`** and **`ServiceID[name=GoogleTalkBotService,vendor=mobicents,version=1.0]`**, would be **`javax.slee.SbbNotification[service=ServiceID[name=GoogleTalkBotService, vendor=mobicents,version=0.1], sbb=SbbID[name=GoogleTalkBotSbb,vendor=mobicents, version=0.1]].GoogleTalkBotSbb`** (without the spaces or breaks), which means a log4j category defining its level as **`DEBUG`** could be:

```
<category
 name="javax.slee.SbbNotification[service=ServiceID[name=GoogleTalkBotService,
 vendor=mobicents,version=0.1],sbb=SbbID[name=GoogleTalkBotSbb,
 vendor=mobicents,version=0.1]]">
    <priority value="DEBUG" />
</category>
```

The relation of JAIN SLEE **`tracers`** and log4j **`loggers`** goes beyond log4j showing tracer's messages, changing the tracer's log4j logger **`effective level`** changes the tracer level in SLEE, and vice-versa. Since JAIN SLEE tracer levels differ from log4j logger levels a mapping is needed:

Table 5.1. Mapping JAIN SLEE Tracer Levels with Apache Log4j Logger Levels

Tracer Level	Logger Level
OFF	OFF
SEVERE	ERROR
WARNING	WARN
INFO	INFO
CONFIG	INFO
FINE	DEBUG
FINER	DEBUG
FINEST	TRACE

High Availability and Fault Tolerance

JAIN SLEE supports clustering, whether it is simple high availability or complete fault tolerance support. This is achieved through the replication of the container state. It also exposes a clustering API for JAIN SLEE Resource Adaptors components, which live outside the container.

6.1. JAIN SLEE Cluster

JAIN SLEE clustering is defined by the server profile selected:

default
: This configuration provides no state replication. It is best used for pure high availability cluster setups.

all
: This configuration provides state replication, and is useful for fault tolerance cluster setups. All activity context and SBB entity data is replicated across the cluster.

  Timers are failed over. All timers running in a node that leaves the cluster, or fails, are recreated in another node.

  Events are not failed over due to performance restraints. This means that an event fired will be lost if its cluster node fails before it is routed.

Important

JAIN SLEE reuses the JBoss Enterprise Application Platform clustering framework, and if all nodes of a cluster are in the same network then the underlying clustering will automatically handle the discovery of new cluster nodes and join these to the cluster. For more complicated setups, refer to the JBoss Enterprise Application Platform clustering documentation.

6.1.1. Managing Components in JAIN SLEE Clusters

JAIN SLEE clustering does not require special components management. Components can be deployed and undeployed in all cluster modes, including fault tolerance setups, and the cluster will handle the operation correctly. However, there are certain behaviours in fault tolerance setups to be aware of:

`Service Activation`
: JAIN SLEE Service started events are only fired on the first cluster node started.

`Service Deactivation`
: Only the last node will force the removal of the service's SBB entities.

`Resource Adaptor Entity Deactivation`
: Only the last node will force the removal of all its activities.

6.2. Fault Tolerant Resource Adaptor API

JAIN SLEE Resource Adaptors exist on the boundary between the container and the underlying protocol. The specification contract requires the RA object to implement the **`javax.slee.resource.ResourceAdaptor`** interface. This interface defines callbacks, which SLEE uses to interact with the RA, including one to provide the

`javax.slee.resource.ResourceAdaptorContext` object. The Resource Adaptor Context provides RA object facilities to interact with SLEE.

The JAIN SLEE 1.1 RA API is a major milestone, standardizing RA and JSLEE contract. Unfortunately it is not enough, it misses an API for clustering, and that is critical for a RA working with a clustered JAIN SLEE. Contarct does not define any fault tolerant data source nor callbacks to inform on cluster state.

The **Fault Tolerant RA API** extends the JAIN SLEE 1.1 RA API, providing missing features related to clustering, and it does it in a very familiar way, anyone who develops a JAIN SLEE 1.1 RA easily uses our proprietary API.

6.2.1. The Fault Tolerant Resource Adaptor Object

The core of the Fault Tolerant RA API is the **`org.mobicents.slee.resource.cluster.FaultTolerantResourceAdaptor`** interface. It is intended to be used instead of the **`javax.slee.resource.ResourceAdaptor`** interface from the JAIN SLEE 1.1 Specification.

The FaultTolerant interface provides three new callback methods used by the container:

`setFaultTolerantResourceAdaptorContext(FaultTolerantResourceAdaptorContext context)`
: This method provides the RA with the **`org.mobicents.slee.resource.cluster.FaultTolerantResourceAdaptorContext`** object, which gives access to facilities related with the cluster. This method is invoked by SLEE right after invoking **`setResourceAdaptorContext( ResourceAdaptorContext context)`** from JAIN SLEE 1.1 specs.

`unsetFaultTolerantResourceAdaptorContext()`
: This method indicates that the RA should remove any references it has to the FaultTolerantResourceAdaptorContext, as it is not valid anymore. The method is invoked by SLEE before invoking **`unsetResourceAdaptorContext()`** from JAIN SLEE 1.1 specs.

`failOver(K key)`
: Callback from SLEE when the local RA was selected to recover the state for a replicated data key, which was owned by a cluster member that failed. The RA may then restore any runtime resources associated with such data.

`dataRemoved(K key)`
: Optional callback from SLEE when the replicated data key was removed from the cluster, this may be helpful when the local RA maintains local state.

6.2.2. The Fault Tolerant Resource Adaptor Context And Replicated Data Sources

Cluster RA context follows the contract of JAIN SLEE 1.1 specification interface **`javax.slee.resource.ResourceAdaptorContext`**. It gives access to facilities for RA to run in clustered environment.

The cluster contract is defined in: **`org.mobicents.slee.resource.cluster.FaultTolerantResourceAdaptorContext`**. It provides critical information, such as if SLEE is running in local mode (not clustered), if it is the head/master member of the cluster, and what the members of the cluster are.

It also provides two data sources to replicate data in cluster:

`ReplicatedData`
: A container for serializable data, which is replicated in the SLEE cluster, but don't require any failover.

`ReplicatedDataWithFailover`
: A **`ReplicatedData`** which requires fail over callbacks, this means, that for all data stored here, when a cluster member goes down, the SLEE in another cluster member will invoke the **`failOver(Key k)`** callback in the Fault Tolerant RA object.

When retrieved from the context through a boolean parameter, both types of **`ReplicatedData`** can activate the callback on the **`FaultTolerantResourceAdaptor`** which indicates that a specific data was removed from the cluster remotely.

6.3. Resource Adaptor Activity Replication

The Resource Adaptor API includes an optional component named **`javax.slee.resource.Marshaler`**, which is responsible, besides other functions, for converting Activity Handles to byte arrays and vice-versa. Also relevant, the Resource Adaptor, when starting activities, may provide a flag indicating that the container may marshall the activity (using the Marshaler). In case of a container cluster with data replication, if an activity is to be replicated then the Marshaler must be provided and the activity flags must activate the flag MAY_MARSHALL, otherwise the activity is not replicated and if a node fails all its activities are removed from the container cluster.

Note

The activity replication doesn't mean that the activity object is replicated by any means, only the related Activity Handle. The Resource Adaptor must use the Fault Tolerant RA API or its own means to replicate any additional data to support that presence of the activity in all nodes of the cluster. Usage of the Fault Tolerant RA API is recommended since it reuses the clustering setup of the container.

Firing Events from Java EE Applications

7.1. SLEE Connection Factory

The JAIN SLEE specification includes an API for interaction with JAIN SLEE container, with the interface **`javax.slee.connection.SleeConnectionFactory`** upfront, which allows external applications, such as an EJB, to create connections to a specific JAIN SLEE instance, and use that to fire events.

JAIN SLEE provides two different implementations of the API, one for access in the same JVM, another for remote access. Both implementations expose the SLEE Connection Factory in the local JNDI tree, thus the same code is used by the application independently of the implementation used.

7.1.1. Local SLEE Connection Factory

JAIN SLEE container instances always expose the local SLEE Connection Factory in its JVM, which means that an external application running in the same JVM doesn't need any additional tools or setup to use it.

7.1.2. Remote SLEE Connection Factory

JAIN SLEE includes a tool, named Remote SLEE Connection Tool, which is a JCA connector that can be deployed in any Java EE application server. Once deployed this connector installs the factory into JNDI, which communicates with the remote SLEE container through RMI.

JBoss Communications Platform Remote SLEE Connection Tool is bundled in all JBCP JAIN SLEE releases, in the tools/remote-slee-connection directory, including its own documentation.

7.1.3. Using SLEE Connection Factory

Below is example code which retrieves the SLEE Connection Factory and uses it to fire an event into the JAIN SLEE container. The code is the same whether the SLEE container is in the same JVM or not.

```


// retrieves JNDI context
InitialContext context = new InitialContext();

// retrieves the connection factory from JNDI
SleeConnectionFactory factory = (SleeConnectionFactory) context.lookup("java:/
MobicentsConnectionFactory");

// creates a connection to the SLEE container
SleeConnection connection = factory.getConnection();

// creates the activity handle which will be used to fire the event
ExternalActivityHandle handle = connection.createActivityHandle();

// retrieves the event type ID
EventTypeID eventTypeID = connection.getEventTypeID("CustomEvent", "...", "1.0");

// creates the event object
CustomEvent eventObject = new CustomEvent();
```

```
// fires the event in the remote SLEE container
connection.fireEvent(eventObject, eventTypeID, handle, null);

// closes the connection
connection.close();
```

JAIN SLEE 1.1 Extensions

JBoss Communications Platform JAIN SLEE exposes proprietary extensions to the JAIN SLEE 1.1 specification, to allow the development of easier or more powerful application code.

The extensions were discussed among multiple JAIN SLEE vendors, and should become part of the standard in next revision, but there is no guarantee that portability won't be lost when using those.

The extensions source code is available in the Mobicents Community SVN, specifically at *http://mobicents.googlecode.com/svn/tags/servers/jain-slee/2.x.y/api/extensions/1.0.0.BETA1*. The setup for the source project in Eclipse IDE is similar to the container core, as seen in *Section 9.4, “Setting JAIN SLEE Source Code Projects in Eclipse IDE”*.

Java archives (JARs) with compiled classes, javadocs and sources are available in the JBoss Maven Repository at *http://repository.jboss.org/nexus/content/groups/public/org/mobicents/servers/jainslee/api/jain-slee-11-ext/1.0.0.BETA1/*

8.1. SbbContext Extension

This extension to JAIN SLEE 1.1 introduces **`org.mobicents.slee.SbbContextExt`** interface, which extends **`javax.slee.SbbContext`** with methods to retrieve SLEE factories and facilities, avoiding the usage of JNDI context.

```


package org.mobicents.slee;

import javax.slee.ActivityContextInterface;
import javax.slee.Sbb;
import javax.slee.SbbContext;
import javax.slee.facilities.ActivityContextNamingFacility;
import javax.slee.facilities.AlarmFacility;
import javax.slee.facilities.TimerFacility;
import javax.slee.nullactivity.NullActivityContextInterfaceFactory;
import javax.slee.nullactivity.NullActivityFactory;
import javax.slee.profile.ProfileFacility;
import javax.slee.profile.ProfileTableActivityContextInterfaceFactory;
import javax.slee.resource.ResourceAdaptorTypeID;
import javax.slee.serviceactivity.ServiceActivityContextInterfaceFactory;
import javax.slee.serviceactivity.ServiceActivityFactory;

public interface SbbContextExt extends SbbContext {

 public Object getActivityContextInterfaceFactory(
   ResourceAdaptorTypeID raTypeID) throws NullPointerException,
   IllegalArgumentException;

 public ActivityContextNamingFacility getActivityContextNamingFacility();

 public AlarmFacility getAlarmFacility();

 public NullActivityContextInterfaceFactory getNullActivityContextInterfaceFactory();

 public NullActivityFactory getNullActivityFactory();

 public ProfileFacility getProfileFacility();

 public
 ProfileTableActivityContextInterfaceFactory getProfileTableActivityContextInterfaceFactory();

 public Object getResourceAdaptorInterface(ResourceAdaptorTypeID raTypeID,
   String raEntityLink) throws NullPointerException,
   IllegalArgumentException;

 public ServiceActivityContextInterfaceFactory getServiceActivityContextInterfaceFactory();

 public ServiceActivityFactory getServiceActivityFactory();

 public TimerFacility getTimerFacility();
}
```

The **`getActivityContextInterfaceFactory(ResourceAdaptorTypeID)`** method:
: Retrieves the ActivityContextInterface factory for the Resource Adaptor Type with the specified ID.

The **`getActivityContextNamingFacility()`** method:
: Retrieves the Activity Context Naming Facility.

The **`getAlarmFacility()`** method:
: Retrieves the Alarm Facility.

The **`getNullActivityContextInterfaceFactory()`** method:
: Retrieves the Null Activity Context Interface Factory.

The **`getNullActivityFactory()`** method:
: Retrieves the Null Activity Factor.

The **`getProfileFacility()`** method:
: Retrieves the Profile Facility.

The **`getProfileTableActivityContextInterfaceFactory()`** method:
: Retrieves the Profile Table Activity Context Interface Factory.

The **`getResourceAdaptorInterface(ResourceAdaptorTypeID,String)`** method:
: Retrieves the interface to interact with a specific Resource Adaptor entity, identified by both the entity link name and the Resource Adaptor Type ID.

The **`getServiceActivityContextInterfaceFactory()`** method:
: Retrieves the Service Activity Context Interface Factory.

The **`getServiceActivityFactory()`** method:
: Retrieves the Service Activity Factory.

The **`getTimerFacility()`** method:
: Retrieves the Timer Facility.

8.2. ProfileContext Extension

This extension to JAIN SLEE 1.1 introduces **`org.mobicents.slee.ProfileContextExt`** interface, which extends **`javax.slee.ProfileContext`** with methods to retrieve SLEE alarm facility, avoiding the usage of JNDI context.

```


package org.mobicents.slee;

import javax.slee.facilities.AlarmFacility;
import javax.slee.profile.Profile;
import javax.slee.profile.ProfileContext;

public interface ProfileContextExt extends ProfileContext {
 public AlarmFacility getAlarmFacility();

}
```

The **`getAlarmFacility()`** method:
: Retrieves the Alarm Facility.

8.3. ActivityContextInterface Extension

This simple extension to JAIN SLEE 1.1 introduces **`org.mobicents.slee.ActivityContextInterfaceExt`** interface, which extends **`javax.slee.ActivityContextInterface`** with methods to retrieve the timers and names bound to the ACI.

```


package org.mobicents.slee;

import javax.slee.ActivityContextInterface;
import javax.slee.facilities.TimerID;

public interface ActivityContextInterfaceExt extends ActivityContextInterface {

 public TimerID[] getTimers();

 public String[] getNamesBound();

}
```

The **`getTimers()`** method:
: Retrieves the IDs of timers currently set which are related to the ACI.

The **`getNamesBound()`** method:
: Retrieves the names currently bound to the ACI.

8.4. Library References Extension

JAIN SLEE 1.1 standard introduced the Library component, a wrapper for a set of jars and/or classes to be referenced and used by other components types, such as SBBs.

The usage of the standard Library component is very limited, each Library can only refer other Library components. Due to this limitation a developer may not be able to include classes in a Library that depend, just as example, on Resource Adaptor Type interfaces, unless of course those interfaces are in a Library too.

This extension allows libraries to reference other component types, which the developer should use when the classes in the Library need to use classes from that component, by simply extending the JAIN SLEE 1.1 Library Jar XML descriptor.

8.4.1. Extended Library Jar XML Descriptor DTD

The DTD document changes for the extended library jar XML descriptor:

```

<!--
The library element defines a library.  It contains an optional description
about the library, the name, vendor, and version of the library being defined,
zero or more references to any other components that this library
```

```
depends on, and information about zero or more jar files that contain
prepackaged classes and resources to be included with the library.

The classes and resources for a library are the sum total of the classes and
resources contained in:
- the library component jar itself (if any)
- the library jars specified by the jar elements (if any)

All these classes are loaded by the same classloader.

Used in: library-jar
-->
<!ELEMENT library (description?, library-name, library-vendor, library-version,
 event-type-ref*, library-ref*, profile-spec-ref*, resource-adaptor-type-ref*,
 sbb-ref*, jar*)>

<!--
The event-type-ref element identifies an event type that the library classes depend.
It contains the name, vendor,and version of the event type.

Used in: library
-->
<!ELEMENT event-type-ref (event-type-name, event-type-vendor,
          event-type-version)>

<!--
The event-type-name element contains the name of an event type referred by
the library.

Used in: event-type-ref

Example:
    <event-type-name>
        javax.csapi.cc.jcc.JccCallEvent.CALL_CREATED
    </event-type-name>
-->
<!ELEMENT event-type-name (#PCDATA)>

<!--
The event-type-vendor element contains the vendor of an event type referred by
the library

Used in: event-type-ref

Example:
    <event-type-vendor>javax.csapi.cc.jcc</event-type-vendor>
-->
<!ELEMENT event-type-vendor (#PCDATA)>

<!--
The event-type-version element contains the version of an event type referred by
the library

Used in: event-type-ref

Example:
    <event-type-version>1.1</event-type-version>
-->
<!ELEMENT event-type-version (#PCDATA)>

<!--
The profile-spec-ref element identifies an profile specification that the library
classes depend. It contains an optional description about the reference, and the
name, vendor, and version of the referenced profile specification.

Used in: library
-->
```

```
<!ELEMENT profile-spec-ref (description?, profile-spec-name,
          profile-spec-vendor, profile-spec-version)>

<!--
The profile-spec-name element contains the name of a profile specification
component.

Used in: profile-spec-ref

Example:
    <profile-spec-name>AddressProfileSpec</profile-spec-name>
-->
<!ELEMENT profile-spec-name (#PCDATA)>

<!--
The profile-spec-vendor element contains the vendor of a profile specification
component.

Used in: profile-spec-ref

Example:
    <profile-spec-name>javax.slee</profile-spec-name>
-->
<!ELEMENT profile-spec-vendor (#PCDATA)>

<!--
The profile-spec-version element contains the version of a profile
specification component.

Used in: profile-spec-ref

Example:
    <profile-spec-version>1.0</profile-spec-version>
-->
<!ELEMENT profile-spec-version (#PCDATA)>

<!--
The resource-adaptor-type-ref element identifies an resource adaptor type that the
library classes depend. It contains the name, vendor,and version of the RA type.

Used in: library
-->
<!ELEMENT resource-adaptor-type-ref (resource-adaptor-type-name,
     resource-adaptor-type-vendor, resource-adaptor-type-version)>

<!--
The resource-adaptor-type-name element contains the name of a resource
adaptor type component referred by the library.

Used in: resource-adaptor-type-ref

Example:
    <resource-adaptor-type-name>JCC</resource-adaptor-type-name>
-->
<!ELEMENT resource-adaptor-type-name (#PCDATA)>

<!--
The resource-adaptor-type-vendor element contains the vendor of a resource
adaptor type component referred by the library.

Used in: resource-adaptor-type-ref

Example:
    <resource-adaptor-type-vendor>
        javax.csapi.cc.jcc
    </resource-adaptor-type-vendor>
-->
```

```
<!ELEMENT resource-adaptor-type-vendor (#PCDATA)>

<!--
The resource-adaptor-type-version element contains the version of a resource
adaptor type component referred by the library.

Used in: resource-adaptor-type-ref

Example:
    <resource-adaptor-type-version>1.1</resource-adaptor-type-version>
-->
<!ELEMENT resource-adaptor-type-version (#PCDATA)>

<!--
The sbb-ref element identifies an SBB that the library classes depend.
It contains the name, vendor,and version of the SBB.

Used in: library
-->
<!ELEMENT sbb-ref (sbb-name, sbb-vendor,
          sbb-version)>

<!--
The sbb-name element contains the name of a SBB component referred
 by the library.


Used in: sbb-ref

Example:
    <sbb-name>MySbb</sbb-name>
-->
<!ELEMENT sbb-name (#PCDATA)>

<!--
The sbb-vendor element contains the vendor of a SBB component referred
 by the library.

Used in: sbb-ref

Example:
    <sbb-vendor>My Company, Inc.</sbb-vendor>
-->
<!ELEMENT sbb-vendor (#PCDATA)>

<!--
The sbb-version element contains the version of a SBB component referred
 by the library.

Used in: sbb-ref

Example:
    <sbb-version>1.0</sbb-version>
-->
<!ELEMENT sbb-version (#PCDATA)>

<!--
The ID mechanism is to allow tools that produce additional deployment
information (ie. information beyond that contained by the standard SLEE
deployment descriptors) to store the non-standard information in a separate
file, and easily refer from those tools-specific files to the information in
the standard deployment descriptor.  The SLEE architecture does not allow the
tools to add the non-standard information into the SLEE-defined deployment
descriptors.
-->
<!ATTLIST library-jar id ID #IMPLIED>
<!ATTLIST description id ID #IMPLIED>
```

```
<!ATTLIST library id ID #IMPLIED>
<!ATTLIST library-name id ID #IMPLIED>
<!ATTLIST library-vendor id ID #IMPLIED>
<!ATTLIST library-version id ID #IMPLIED>
<!ATTLIST event-type-ref id ID #IMPLIED>
<!ATTLIST event-type-name id ID #IMPLIED>
<!ATTLIST event-type-vendor id ID #IMPLIED>
<!ATTLIST event-type-version id ID #IMPLIED>
<!ATTLIST library-ref id ID #IMPLIED>
<!ATTLIST profile-spec-ref id ID #IMPLIED>
<!ATTLIST profile-spec-name id ID #IMPLIED>
<!ATTLIST profile-spec-vendor id ID #IMPLIED>
<!ATTLIST profile-spec-version id ID #IMPLIED>
<!ATTLIST resource-adaptor-type-ref id ID #IMPLIED>
<!ATTLIST resource-adaptor-type-name id ID #IMPLIED>
<!ATTLIST resource-adaptor-type-vendor id ID #IMPLIED>
<!ATTLIST resource-adaptor-type-version id ID #IMPLIED>
<!ATTLIST sbb-ref id ID #IMPLIED>
<!ATTLIST sbb-name id ID #IMPLIED>
<!ATTLIST sbb-vendor id ID #IMPLIED>
<!ATTLIST sbb-version id ID #IMPLIED>
<!ATTLIST jar id ID #IMPLIED>
<!ATTLIST jar-name id ID #IMPLIED>
<!ATTLIST security-permissions id ID #IMPLIED>
<!ATTLIST security-permission-spec id ID #IMPLIED>
```

This full DTD is available at *http://mobicents.org/slee/dtd/slee-library-jar_1_1-ext.dtd*

8.4.2. Extended Library Jar XML Descriptor Example

The following XML descriptor examples the definition of references to JAIN SLEE 1.1 Component types other than Library

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE library-jar PUBLIC
  "-//Sun Microsystems, Inc.//DTD JAIN SLEE Ext Library 1.1//EN"
  "http://mobicents.org/slee/dtd/slee-library-jar_1_1-ext.dtd">

<library-jar>
 <library>
  <library-name>extended-library-example</library-name>
  <library-vendor>com.redhat</library-vendor>
  <library-version>1.0</library-version>

  <event-type-ref>
   <event-type-name>ExampleX</event-type-name>
   <event-type-vendor>com.redhat</event-type-vendor>
   <event-type-version>1.0</event-type-version>
  </event-type-ref>

  <library-ref>
   <library-name>LibraryX</library-name>
   <library-vendor>com.redhat</library-vendor>
   <library-version>1.0</library-version>
  </library-ref>

  <profile-spec-ref>
   <profile-spec-name>ProfileX</profile-spec-name>
   <profile-spec-vendor>com.redhat</profile-spec-vendor>
   <profile-spec-version>1.0</profile-spec-version>
  </profile-spec-ref>

  <resource-adaptor-type-ref>
```

```
    <resource-adaptor-type-name>ResourceAdaptorTypeX</resource-adaptor-type-name>
    <resource-adaptor-type-vendor>com.redhat</resource-adaptor-type-vendor>
    <resource-adaptor-type-version>1.0</resource-adaptor-type-version>
   </resource-adaptor-type-ref>

   <sbb-ref>
    <sbb-name>SbbX</sbb-name>
    <sbb-vendor>com.redhat</sbb-vendor>
    <sbb-version>1.0</sbb-version>
   </sbb-ref>

  </library>
 </library-jar>
```

Important

Note how the DOCTYPE element is set to the extended DTD instead of the standard one.

Advanced Topics

9.1. Class Loading

Each JAIN SLEE Component has its own classloader (**`ComponentClassLoader`**) in the package named **`org.mobicents.slee.container.component.deployment.classloading`**. This classloader sees the component classes contained in the component jar (**`URLClassLoaderDomain`**) by declaring it as the parent classloader, and adding the classes seen by each component it refers. It does not see classes from a component that it does not depend on.

JAIN SLEE defines a class loading domain with the APIs required in the JAIN SLEE 1.1 container (for example, JAIN SLEE, JBDC and JMX). This domain (**`JBoss Microcontainer ClassLoadingDomain`**) imports all classes shared in the JBoss Enterprise Application Platform, and acts like the parent domain for all **`URLClassLoaderDomain`**s, which means that a class imported by a SLEE classloading domain will always be used first.

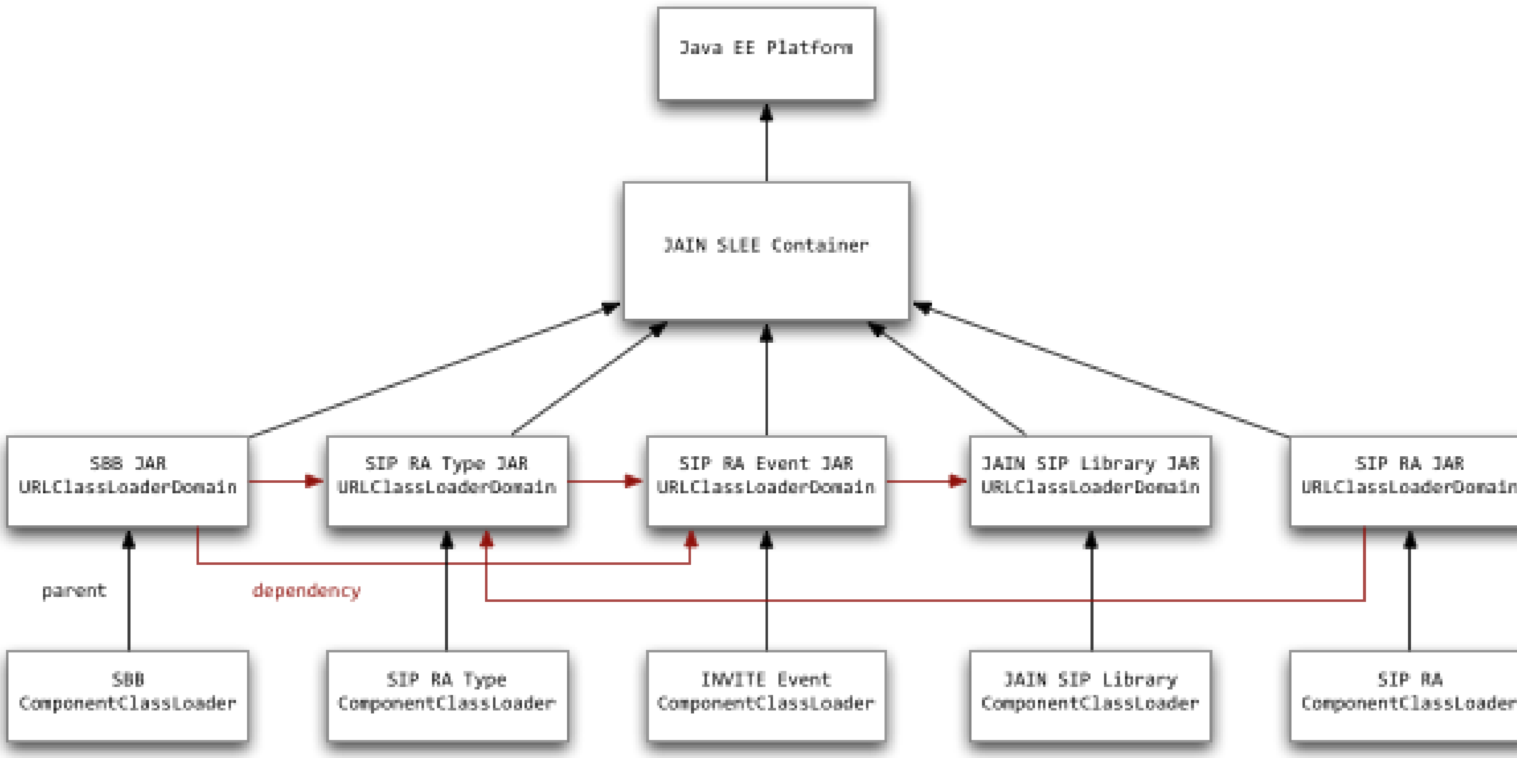
Java EE Platform
JAIN SLEE Container
SBB JAR URLClassLoaderDomain
SIP RA Type JAR URLClassLoaderDomain
SIP RA Event JAR URLClassLoaderDomain
JAIN SIP Library JAR URLClassLoaderDomain
SIP RA JAR URLClassLoaderDomain
parent
dependency
SBB ComponentClassLoader
SIP RA Type ComponentClassLoader
INVITE Event ComponentClassLoader
JAIN SIP Library ComponentClassLoader
SIP RA ComponentClassLoader

Figure 9.1. Classloading example in JBCP JAIN SLEE

- The **`SIP INVITE Event`** component refers to the **`JAIN SIP Library`** in its XML descriptor, and its classloader domain depends on the classloader domain of the JAIN SIP Library.
- The **`SIP RA Type`** component refers to all Events in the SIP RA Event jar in its XML descriptor, and its classloader domain depends on the classloader domain of the SIP Event JAR and inherits its dependencies, including the JAIN SIP library classloading domain.
- The **`SIP RA`** component refers to the **`SIP RA Type`** component in its XML descriptor, and its classloader domain depends on the classloader domain of the SIP RA Type Component jar, and inherits its dependencies. This includes the SIP Event jar and JAIN SIP library classloading domains.

- The **SBB** component refers to the **`SIP RA Type`** component and **`SIP INVITE Event`** in its XML descriptor. Its classloader domain depends on the class loader domain of the SIP RA Type Component jar, and inherits its dependencies; the SIP Event jar and the JAIN SIP library classloading domains. It also depends on the classloader domain of the SIP Event jar.

Important

JBoss Enterprise Application Platform does not see the classes of deployed JAIN SLEE components. This means that if it exports its classes for components that are complemented with Java EE components, the common classes must be deployed on JBoss Enterprise Application Platform, either directly or included in the Java EE component.

9.2. Congestion Control

JAIN SLEE can monitor the memory available in the JVM. In case it drops to a certain level (percentage), new events and/or activity startups are rejected, and at the same time a JAIN SLEE Alarm (which can send JMX notifications) is raised. This feature is called Congestion Control, and the container will turn it off automatically once another available memory level is reached.

If Congestion Control rejects an operation, a **`javax.slee.SleeException`** is thrown. This means that if the feature is to be used, the Resource Adaptors and Applications need to handle such use case, and behave properly.

The type of JAIN SLEE Alarm raised is **org.mobicents.slee.management.alarm.congestion**.

Congestion Control is turned off by default.

9.2.1. Congestion Control Configuration

The Congestion Control feature is configured through an XML file or through a JMX MBean. Changes applied through JMX are not persisted, and once the container is restarted the configuration will revert to the one in the XML file.

9.2.1.1. Congestion Control Persistent Configuration

Configuration is done through a XML descriptor for each *Section 3.1, “Server Profiles”*. The XML file is named **`jboss-beans.xml`** and is located at ***`$JBCP_ROOT`*`/server/`*`$PROFILE`*`/deploy/mobicents-slee/META-INF`**.

The configuration is exposed a JBoss Microcontainer Bean:

```
<bean name="Mobicents.JAINSLEE.CongestionControlConfiguration"
 class="org.mobicents.slee.container.management.jmx.CongestionControlConfiguration">
 <annotation>@org.jboss.aop.microcontainer.aspects.jmx.JMX(name=
  "org.mobicents.slee:name=CongestionControlConfiguration",exposedInterface=
  org.mobicents.slee.container.management.jmx.CongestionControlConfigurationMBean.class,
  registerDirectly=true)</annotation>
 <property name="periodBetweenChecks">0</property>
 <property name="minFreeMemoryToTurnOn">10</property>
 <property name="minFreeMemoryToTurnOff">20</property>
 <property name="refuseStartActivity">true</property>
 <property name="refuseFireEvent">false</property>
</bean>
```

Table 9.1. JAIN SLEE Congestion Control Bean Configuration

Property Name	Property Type	Description
periodBetweenChecks	int	The available memory level is checked periodically, this property defines the period, in seconds, between these checks, and if set to 0 turns off the Congestion Control feature.
minFreeMemoryToTurnOn	int	This property defines the minimum free memory percentage, which if reached turns ON the Congestion Control feature.
minFreeMemoryToTurnOff	int	This property defines the minimum free memory percentage, which if reached turns OFF the Congestion Control feature. This value should be considerably higher than minFreeMemoryToTurnOn, otherwise the feature may be turning on and off all the time.
refuseStartActivity	boolean	If true and the Congestion Control feature is ON, the container rejects activity startups, no matter it's a request from a Resource Adaptor or SBB.
refuseFireEvent	boolean	If true and the Congestion Control feature is ON, the container rejects the firing of events, no matter it's a request from a Resource Adaptor or SBB.

9.2.1.2. Congestion Control JMX Configuration

Through JMX, the Congestion Control feature configuration can be changed with the container running. These configuration changes are not persisted.

The JMX MBean which can be used to change the Congestion Control configuration is named **org.mobicents.slee:name=CongestionControlConfiguration**, and provides getters and setters to change each property defined in the persistent configuration. The JMX Console can be used to use this MBean, see *Section 4.3.1, “JMX Console”*.

9.3. JAIN SLEE 1.1 Profiles JPA Mapping

As mentioned in the containers configuration section, JBoss Communications Platform JAIN SLEE uses JPA to store all JAIN SLEE 1.1 Profiles, and in mentioned section it was explained how to define which JPA / Hibernate data source. In this section more details are provided about how JAIN SLEE 1.1 Profiles are mapped to a JPA datasource schema.

9.3.1. Profile Specification JPA Tables And columns

For each Profile Specification, at least one JPA Table is created, and is named **`SLEE_PE_`** concatenated with the Profile CMP interface simple name (**`obtained as java.lang.Class.getSimpleName()`**), then _, and finally the absolute value of the **`hashCode()`** method of the javax.slee.ComponentID of the Profile Specification.

This table has a primary key composed by the profile name and profile table name, and a column for each attribute of the Profile Specification CMP, except for those of array type. Those columns are named **`C`**, concatenated with the cmp attribute name.

For each Profile CMP attribute of *`array`* type, a join table is created, and is named **`SLEE_PEAAV_`** concatenated with the Profile CMP interface simple name (**`obtained as java.lang.Class.getSimpleName()`**), then _, then the absolute value of **`hashCode()`** method of the javax.slee.ComponentID of the Profile Specification, and finally the CMP attribute name. This table has a generated primary key column named **`ID`**, the foreign key, and two columns to store the CMP attribute value:

`SERIALIZABLE`
: Used to store the value if its type does not allow it to be converted to a String.

`STRING`
: Used when the CMP attribute type can be converted to a **`java.lang.String`**, for instance an **`Integer`**.

9.3.2. Profile Specification JPA Datasource

Unless configured manually, JBoss Communications Platform JAIN SLEE uses the default datasource of JBoss Enterprise Application Platform. Please refer to its documentation to learn about it.

9.4. Setting JAIN SLEE Source Code Projects in Eclipse IDE

The JAIN SLEE Core, each RA, and each example, are worked out with separated Eclipse IDE Projects.

There are two alternatives to set up a specific project:

Procedure 9.1. Via Command Line

1. In the checked out directory of the project, and with Eclipse IDE closed, open a terminal.

2. Run the following:

   ```
   ]mvn mobicents:eclipse
   mvn -Declipse.workspace=YOUR_RELATIVE_PATH_TO_ECLIPSE_WORKSPACE eclipse:add-maven-repo
   ```

3. Install M2Eclipse if you want to do maven builds within Eclipse.

Procedure 9.2. With Eclipse IDE

- Install the M2Eclipse plugin and use "Import..." and subselect the "Maven Projects" feature.

Important

Ensure the "Resolve Workspace projects" and "Separate projects for modules" in the "Advanced" options on the bottom of the window are turned off. If the project is large, such as the JAIN SLEE Core, M2Eclipse may be a considerable slower option, due to dynamic Maven2 Dependency Management.

Appendix A. Revision History

Revision 4.0	**Fri Feb 25 2011**	**Tom Wells** *twells@redhat.com*

JBCP 5.1 GA documentation

Revision 3.0	**Fri Oct 22 2010**	**Tom Wells** *twells@redhat.com*

JBCP 5.x GA documentation

Revision 2.0	**Fri Apr 30 2010**	**Tom Wells** *twells@redhat.com*

Development of JAIN SLEE 5.x Product User Guide.

Revision 1.0	**Tue Dec 22 2009**	**Eduardo Martins** *emartins@redhat.com*

Creation of the JBCP JAIN SLEE 5.x User Guide.

Index